A Robotic PlantCare System

Kevin Sikorski, University of Washington

IRS-TR-03- 006

May 2003

Intel Research

A Robotic PlantCare System

Kevin Sikorski, University of Washington

In fulfillment of the Master's Degree Project Requirement

May 14, 2003
Advisor: Dieter Fox

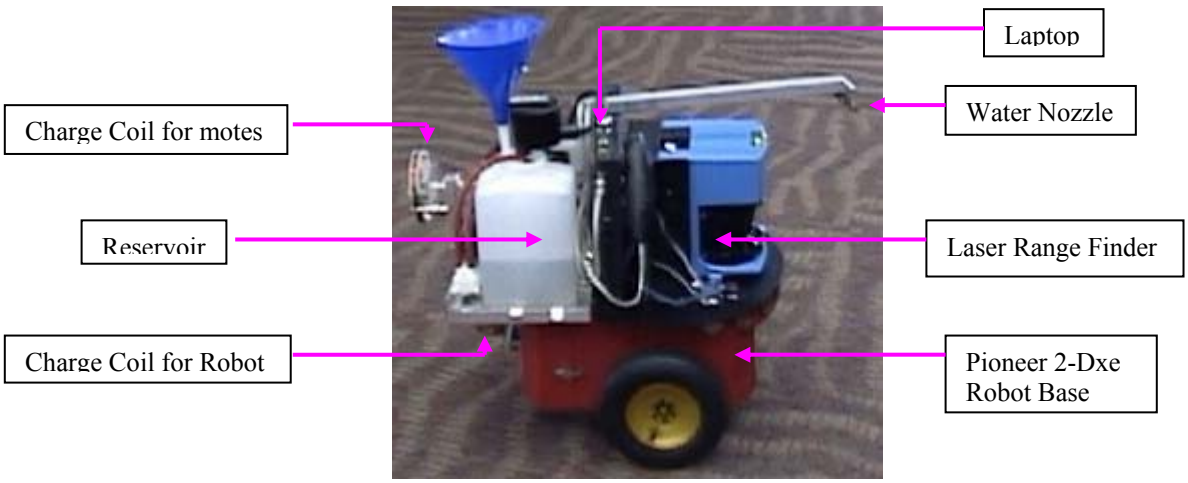
Introduction

Ubiquitous computing is beginning to generate interest in the computer world, and new applications for this technology are being found daily. One facet of ubiquitous computing is the concept of distributed sensor networks: groups of sensors that communicate with each other for tasks such as monitoring inventory (RFID tags and infrastructure in a convenience store), security (security cameras in a warehouse), or watching environmental conditions (a series of temperature sensors in a greenhouse). One major obstacle blocking the adoption of distributed sensor networks is the issue of maintenance. Installing infrastructure to supply power to the sensors would be a robust solution, always keeping them powered, but it is an expensive option. On the other side of the spectrum, one could simply use disposable battery packs to power the sensors. However, these battery packs would have be tested and replaced regularly, a labor intensive process if done by hand. Combining robotics with ubiquitous computing offers the best of both worlds: the immediate, convenient access to technology and distributed sensing, without the requirements for frequent manual (human) maintenance.

The PlantCare project was created with the intention to demonstrate this concept. A number of office plants were kept inside the Intel Research Lablet in Seattle, Washington. Each plant was outfitted with a sensor, equipped with wireless network access, and equipment to assess the condition of the plant. This information would be sent over the network to a desktop computer. Whenever a plant's condition, such as the moisture content of its soil, would fall out of an acceptable range, the computer could active a robot in the lab. This robot would then locate the plant, water it, and recharge the sensor. Then the robot would automatically return to its maintenance bay, where it would recharge itself, and refill its water supply. In the actual implementation of this project, hardware and time difficulties kept us from programming the robot to recharge the plant's sensor, however, the remainder of the project was created successfully.

Hardware

The PlantCare robot consisted of four main components: the base, the laser range finder, a laptop computer, and the plant-watering/mote-charging apparatus. Throughout the project, we used two separate bases. Our first design incorporated a Pioneer 2-Dx from ActivMedia, one of the primary suppliers for academic research robotics. This base is in its own right a full robot, supplying power in the form of 3 Lead-Acid batteries, two articulated wheels and a caster, and a bank of 8 sonar sensors. The sonars were not used for this project due to their inaccuracy and the extra power they would consume. Our second design used a Pioneer 2-Dxe robot, which is slightly larger than the 2-Dx model and with a different wheel design, but otherwise equivalent. In both designs, we added a rail system on the underside of the robot to aid docking, and an inductive coil in the rear for recharging the robot at its maintenance bay. The laser range finder was manufactured by a SICK, Inc., and is also a standard for robotics research. The laser provides range measurements accurate to ± 0.5cm every half-degree over a 180° spread, at about 5Hz. The range finder is mounted directly on the robot, facing forward. This unit provided the sensor information we used for object detection, localization, and obstacle avoidance. You can see the range finder in the picture below – it is the large blue object that resembles a coffee maker. The laptop computer was mounted behind the laser range finder, and we used an IBM laptop with a 500MHz Intel Pentium III processor, and was equipped with access to the Intel lab's wireless network. The plant-watering/mote-charging apparatus was custom built by Intel employees for this project. This component consisted of a PIC board for control, a half-gallon water reservoir, water pump and nozzle for watering plants, and an inductive coil for recharging motes. The first design of the robot positioned the nozzle for watering plants out the rear of the robot.



However, since the laser range finder was directed forward from the robot, this made docking with plants artificially hard. Our second design positioned the water nozzle out the front, as show in the picture.

The maintenance bay consists of a rectangular Plexiglas box open on one side, fitted over a gentle ramp. Inside the bay is an inductive charge coil for recharging the robot while it is "at home", and system for replenishing the robot's water supply. Two rails are attached to the ramp in a V-shape: these were built to aid the robot in its docking process. The robot docks by driving backwards up the shallow ramp, and threading its rear caster through the space between the two rails. Once the robot has driven far back enough, the complementary rail system on the underside of the robot locks with ramp's rails, forcing the robot to drive directly such that its inductive charge coil is inserted directly into the maintenance bay's charge coil. Once the robot is firmly in position, a switch on the maintenance bay is triggered, starting the inductive charger. This switch also triggers the water replenishment system: a valve opens, allowing water to fall from the maintenance bay's reservoir into the robot's funnel, and into the robot's reservoir. A radar sensor on the side of the bay determines when the robot's reservoir is refilled, and closes the valve. We chose to use a reservoir on the maintenance bay for safety purposes, so that in the event of a system failure, the only water than can be spilled is that in the reservoirs. In a real installation, the maintenance bay could be connected to the building's main plumbing.

The maintenance bay also has a small external component, which houses the electrical equipment associated with powering the bay. This extra component also contains a small air compressor and tubing for forcing air through the coils when the robot charges. Because inductive charging is so inefficient (approximately 20%), a large amount of waste heat is created in the coils. Before the air compressor was added to the system, this excess waste heat was once sufficient to melt the coils, preventing any useful power transfer.

The sensors deployed on the plants used the Berkley Mote design for computation *[hill-2000]*, sensor interfaces, and wireless network. Attached to each mote was a light sensor, temperature sensor, and a hygrometer for measuring the moisture content of the soil. The motes were powered by a battery pack, which was attached to an inductive charge coil, which could be charged by a similar coil attached to the

rear of the robot.

Software - BeeSoft

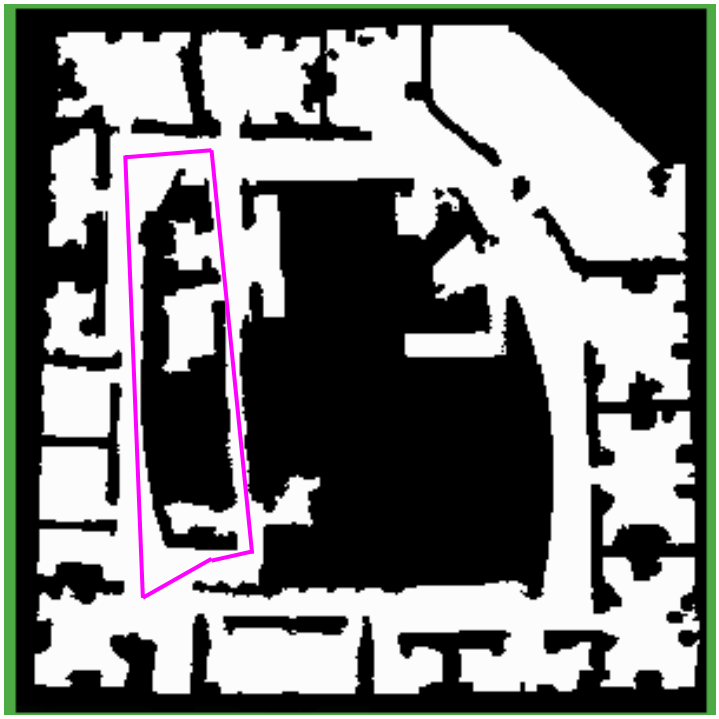
The PlantCare system went through two major versions. The first version utilized an open source, robot-control software suite for Linux called BeeSoft, which includes software for integrating sensor logs into occupancy-grid style maps, path planning, localization, and collision avoidance software. We were also able to leverage a module for remotely controlling the robot via a Java applet, written by Cody Kwok.

The first step was to produce a coherent map for the environment in which the robot would be operating. This map would later be used by the planning module and the localization module; the robot cannot decide on the best path to a goal, or estimate its current location given sensor readings if it does not know the structure of the environment. The map was produced by attaching a joystick to the robot, and manually driving the robot around the lab while logging all sensor readings from the laser range finder and on-board odometer. BeeSoft provides a piece of software that can be process these logs to produce an occupancy grid map for the lab.

While this process may sound trivial, in practice it is as much of an art as a science. Because the lab consists of primarily a large circle, the map building process encounters a correspondence problem when it comes time for the robot to "close the loop". Specifically, map-making programs have difficulty determining when the robot is reaching a place in the map that it has seen before, due to noisy odometry and laser readings. This problem is exacerbated by the fact that a number of parameters corresponding to the program's sensor model must be hand tuned to the environment to produce an accurate map. In addition, it was difficult to find a time when the lab was completely unoccupied. Mapping could be performed while people were still walking around the lab, as the software is capable of screening out transient objects. However, since the objective was to accurately map static features of the environment, most mapping attempts were made at night or on weekends to alleviate the problem of people and objects obscuring important features. Other similar issues were doorways – doors tend to be opened and closed several times during the workday, and so were more easily mapped after hours, when they could be kept open without impacting the regular operation at the lab. Producing a complete and accurate map required that all doors be open, so that the robot could explore all rooms' interiors.

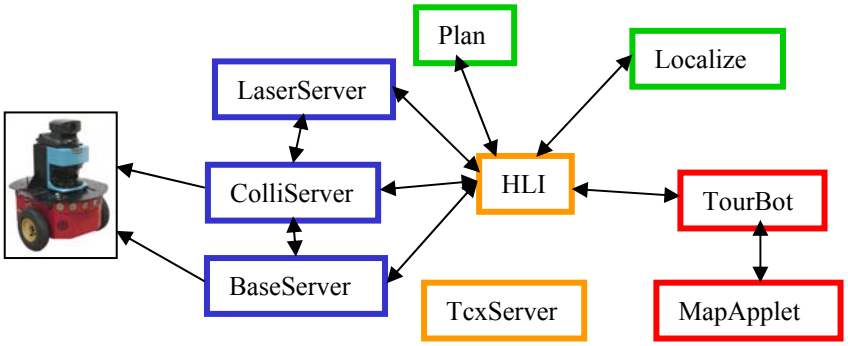
Another major obstacle in building an accurate map of the lab was the fact that it contained surfaces that were hostile to laser reflections. For example, the walls of the main conference room are made of glass plate. This means that laser light sometimes reflects off of the walls as off a mirror; other times, the light just passes through. Thus I had to be sure to map several regions of the lab multiple times, to ensure that such incidental reflections would not be interpreted by the mapping software as a person walking past the robot or some other transient obstacle. After several failed attempts, I found that the best circuit to take around the lab was to start at an arbitrary location on the smaller loop. I then drove the robot around this loop, then around the full lab twice. Then I made a third circuit around the lab, bringing the robot into each cubicle and room. Finally, I made a fourth trip around the lab, this time in the opposite direction. Note that in the map below, the doors to the hardware lab were not open, so the large black region in the lower right was never properly mapped. The large black triangle in the upper right actually corresponds to the balcony, an area where it is unsafe to drive the robot.

The smaller loop is shown in magenta. The starting location is the Northeast corner of this loop. The maintenance bay is located along the inner wall, at the Northwest corner of this loop (not shown in map).



BeeSoft Software Organization

The basic organization of BeeSoft as used in PlantCare is shown below. The blue-colored boxes handle basic sensor interfacing (LaserServer), obstacle avoidance (ColliServer), and low level movement control (BaseServer), the green boxes handle the map-related functions of path-planning (Plan) and localization (Localize), the red boxes handle higher-level control and user interfaces (intended for debugging purposes, as the overall goal was to produce a system that required no true user interface), and the orange boxes handle message passing (HLI – High Level Interface) and allow the different modules to find each other (TcxServer).



Once the map for the Intel lab was produced, the bulk of the effort was spent on coding an efficient docking strategy for both the office plants and the maintenance bay. For this, we settled on a "scripted" approach where the robot performs a specific series of actions to properly dock with its target. We modified the Java Map Applet, allowing the user to execute the docking scripts with a single button press, as well as change movement speeds, drive autonomously to user-specified coordinates, and control the robot via primitive commands such as "move forward 1 meter" or "turn right 30 degrees".

Once the robot received a command from the base station or the user to dock with the maintenance bay, it would:

1. Drive to a human-determined point near the maintenance bay.
2. Drive to the front of the bay.
3. Turn to an absolute heading to face the bay.
4. Execute a 180° turn so that the rear caster would be facing between the ramp rails.
5. Back into the bay at a high rate of speed to drive up the ramp.
6. Back up more slowly until the robot has fully docked.

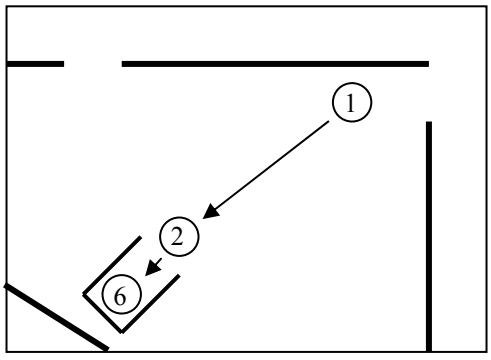


Diagram illustrating the steps taken for the robot to dock with the maintenance by for the BeeSoft implementation. The robot drives to a known point (1). It then drives to another point at the front of the bay (2), properly establishing an appropriate approach vector. It turns around and backs into the bay, eventually arriving at (6).

Docking with office plants was performed in a similar manner. Originally, this approach looked like a reasonable and easy-to-implement solution to docking. However, we soon found that the low-level movement control software was not designed to make accurate movements. Empirically, we found that its accuracy was limited to arriving at a target point within about 30cm. Unfortunately, the space between the two ramp rails to accommodate the caster was only about 6cm wide. Thus, this solution was unreliable, working about 90% of the time at its best, but often only 20% of the time. We considered making modifications to the low-level movement control software, but this would have required a full rewrite of the

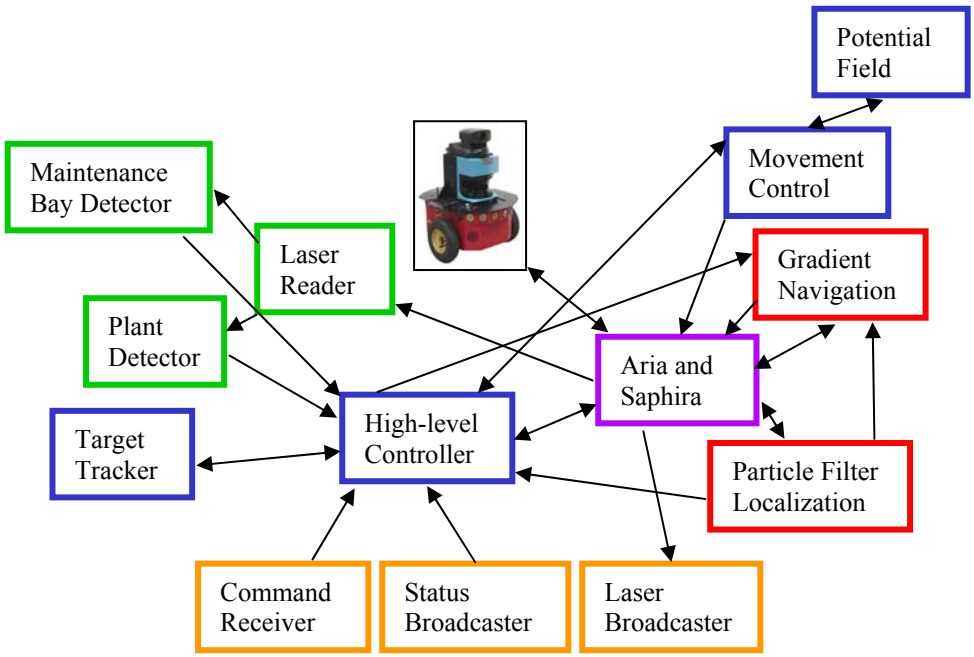
code. Interestingly, the success rate often varied day-to-day – this was partially due to the hard-coded coordinates used for docking. Occasionally, the maintenance bay might be moved an inch or two in cleaning or just daily usage, which would the robot to dock towards point completely unassociated with the maintenance bay. Our second implementation (see below) used a different approach that avoided this problem.

There were other issues with this implementation of the PlantCare project relating to the BeeSoft software. While having the source code and being able to modify it was a significant advantage of BeeSoft, we could also see that it had been worked on by a large number of researchers, using widely varying coding conventions and styles, conflicting message names for interprocess communication, and only very rarely documenting anything. Internal comments, when present, were in multiple languages (English and German). The lack of even README-type files made understanding exactly how the various modules worked together difficult, and there was no central resource for software support. In addition, some modules were buggy – most notably, the path-planning module. Even simple features such as notifying a higher-level control program that the robot has reached its destination had to be tested and debugged before they could be relied on to work. Due to these problems, we eventually abandoned this software package, and moved on to our second implementation of the PlantCare project.

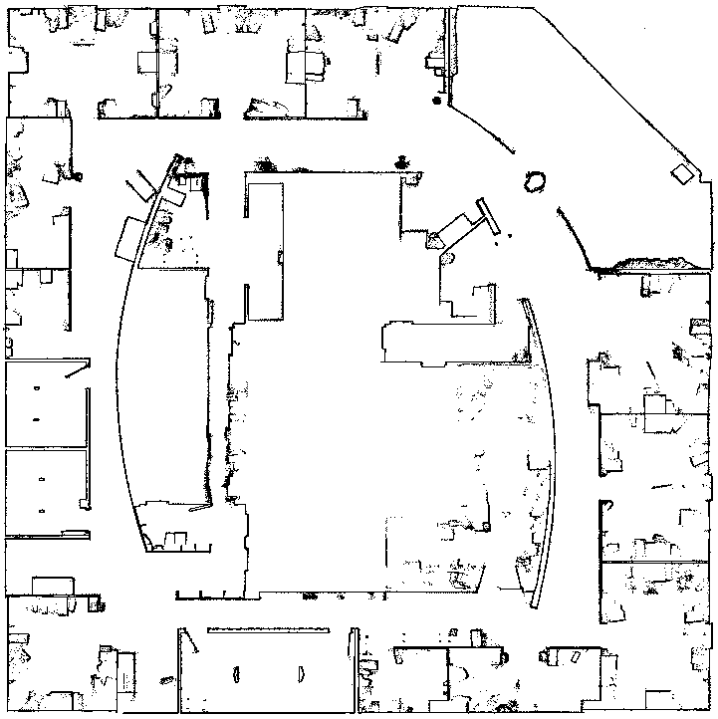
Software: Aria/Saphira

Our second design was built on top of ActivMedia's Aria and Saphira software packages. Aria is an open source robot control package that is moderately well documented and supported, handles basic movement and sensor interpretation, and provides a robust behavior-based framework for writing applications. Saphira is a higher-level, closed source add-on to Aria, which can provide particle-filter-based localization *[Fox-1999]* and navigation and obstacle avoidance via the gradient method [Konolige-2000]. Together, these two packages provide virtually all of the functionality of BeeSoft, with the minor exception of a few debugging utilities and the ability to perform some computation off-board the robot. I was able to write the debugging utilities to fill this vacuum, and Aria's platform-independent socket implementation made it easy to transmit robot state and sensor information to a desktop computer, and commands back to the robot. This meant that off-board computation was also possible. However, the system ran fast enough on our laptop that this was done only for logging, testing, debugging, and demoing purposes.

Below, we illustrate the overall organization of the second PlantCare design. Green boxes indicate high-level feature detection code, red boxes include localization and path planning, blue boxes handle movement and high-level control, orange boxes handle sending state and sensor information to off-board computers for monitoring and debugging purposes, and the purple box contains the vendor-provided Aria and Saphira code.

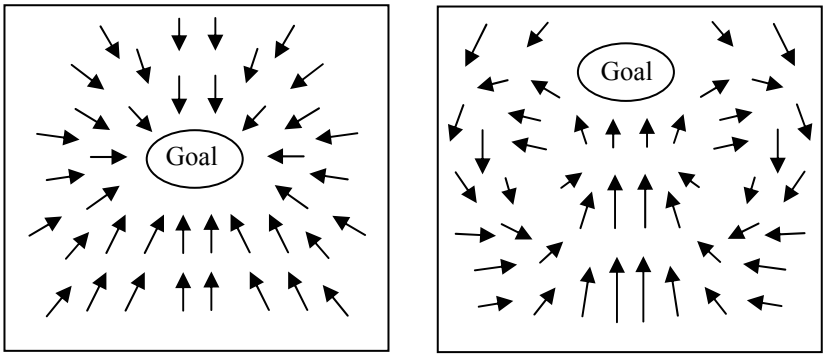


We went through a second map-building phase for Aria and Saphira that was similar to that for BeeSoft. We purchased additional software from ActivMedia, which was capable of integrating sensor logs into a coherent map, and again drove the robot around the lab by joystick logging sensor readings. While the software was imperfect (it crashed on all but one sensor log using all but one laser scan-matching technique – full bug reports were submitted to the authors), we were able to produce a very accurate map of the lab which was used for the remainder of the project, and has since found use in several other localization and tracking projects be worked on at the lab.



A slightly hand-cleaned version of the final map used for the PlantCare project. The automatically-generated map left a large amount of "dust" that clearly was not associated with any real objects in the lab. We also removed movable objects such as chairs and whiteboards.

We settled on a different docking approach for this implementation. Instead of the "scripted" solution, we decided on a behavior-based docking strategy. Behavior-based robotics is a common paradigm in robotics where we consider the robot to be a stateless agent in an environment. The robot's actions are guided by "behaviors": for each behavior, the robot maintains a vector field specifying how it should move for any attainable position in the environment. This vector field can be thought of as a grid, where each point contains a vector, with the direction of the vector indicating in which direction the robot should move, and with the magnitude of the vector indicating how fast the robot should move for that behavior. As an example, a simple "attract" behavior is depicted below. At each time step, the robot looks up its current location relative to the goal in each vector field and sums the vectors. This sum then indicates the direction and speed with which the robot should move for the composition of the given behaviors. This approach is also extremely fast and takes minimal memory, because we never have to compute the composition vectors for the entire field – we only need to compute a vector sum corresponding to the robot's current position in the world.



Left: An example "attract" behavior represented as a vector field.
Right: A simplified version of the "docking" behavior used. Note that if the robot is directly in front of the plant, it drives directly towards the goal. If it is slightly to the side, it tries the center itself. If it is close to the goal but off to the side, the robot turns around and retries its approach.

In order to dock with the maintenance bay, the robot:

1. Drives to a point and orientation from which it can see the maintenance bay.
2. Uses a "docking behavior" to drive to a point a fixed distance from the opening in the ramp rails and with the appropriate orientation.
3. Aligns itself to face directly into bay
4. Performs a 180° turn to align its caster with the rail opening.
5. Drives backwards quickly to climb the ramp.
6. Drives backwards more slowly until it has docked with the inductive charger.

Docking with plants proceeds similarly:

1. Drives to a point and orientation from which it can see the plant pot.
2. Uses a "docking behavior" to drive to a point a fixed distance from the plant and with the appropriate orientation.
3. Aligns itself to face directly towards the center of the pot.
4. Drives forwards or backwards until it is a specified distance away from the plant.

The primary advantage of using a behavior-based method for docking is robustness. The scripted form relied on plants and the maintenance bay being at predetermined locations, and not moving from those locations. In reality, a perfectly static environment is not feasible – even in a laboratory setting, objects are nudged or bumped into by people, even accidentally. Over time, such movement can cause scripted approaches to fail. This reactive strategy is more robust is that the robot need only "see" its target from a hand-crafted point and orientation; once it can see the target, the behavior-based docking field takes over, and guides the robot to the target point accurately.

Working with Aria and Saphira was also much easier than working with BeeSoft. The documentation was not perfect, but was ample, and ActivMedia actively supported its software by fixing bugs that we reported through frequent patches, and answering questions via a mailing list. This software also had the advantage of being able to run all the software on the laptop. We still found it useful to be able to redirect sensor readings to other computers for easy debugging, and fortunately it was easy to write such visualization software using the Fast Light Tool Kit (FLTK) graphics package.

Feature Detection

So far, we have not discussed exactly when mean when we say that the robot "sees" a plant pot or the maintenance bay. For sensing obstacles, we used the laser range finder. Both the plant pot and the bay have a distinctive signature on the range finder – the plant pot shows up as a circle, and the maintenance bay appears as a rectangle with specific dimensions. Each cycle, the robot receives a new laser scan in the form of an array, where each element represents the range at a particular bearing from the robot. The system then extracts the appropriate features from the scan, as well as a value indicating how well the observation fits the expected profile of the feature. We discuss the techniques used for feature extraction below.

Plant Pot Detection

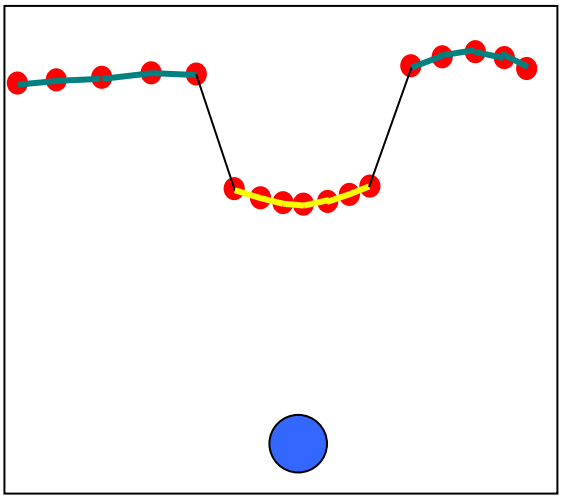
As mentioned above, plant pots show up as circles on the laser range finder. Our first approach at extracting the circles was to apply Hough transforms. Hough transforms are a classic way to find features in a cloud of points by mapping each detected point in sensor-space into another feature space, Hough-space, where features can be extracted trivially *[Iocchi-1999]*. For example, for plant pot detection, we want to extract circles of a preset radius from sensor readings. In this case, Hough-space is a two dimensional grid of accumulators, where the x and y coordinates in this grid indicate the (x,y) location of the center of the circle relative to the robot. We proceed by processing each point returned by the laser range finder in turn. For each point, we "draw" the Hough image of that point in Hough space. In our example, the Hough image is the collection of points in Hough space that could be the center of a circle with the given radius that generated that sensor point, which in this case, is a circle. Because Hough-space is implemented as a grid of accumulators, when "drawing" the circle, we simply increment the accumulators that fall under the point's Hough image. After we have drawn the Hough images of all the

points, we can find the circle that was best represented by the sensor readings by identifying the Hough-space accumulator with the largest value (i.e., lies a fixed radius away from the most points). Because in our environment, the only objects that would appear as 22.5cm radius circles in laser scans were plant pots, we were able to use this approach to modest success.

Hough transforms do exhibit some difficulties. First off, they are computationally intensive: For each point, we must increment a number of accumulators in Hough-space. One can perform simple feature-specific optimizations to the transformation to make it run faster. For example, our version recognized that the laser could not see through plant pots. This means that only half of a circle had to be drawn in Hough-space for each point; for the remainder of the points, the only way that the laser range finder could have seen them would have been for the laser to travel through the pot, which physically impossible. This roughly doubled the execution speed of the detector.

The second major difficulty encountered when working with Hough transforms was the coarseness of the grid used – A very fine grid means more accurate positioning for the centers of detected circles, but requires that more accumulators are accessed, and is a large drain on memory. Conversely, a coarse grid detects features very quickly and requires little memory, but does not detect the positions of features accurately. This meant that the robot could either be programmed to accurately determine the locations of features close by, or to inaccurately find the locations of far away objects. A potential solution to this problem would be to run multiple Hough transforms at varying scales. One could run two transformations per cycle: the first a coarse version detecting the general location of a pot, and the second a finer version for "zooming in"and determining the exact location of the target. A similar work-around that I used in my undergraduate thesis on Sonar-Based Navigation and Feature Detection was to run just one transformation per cycle, but instead to vary the granularity each cycle *[Sikorski-1999]*. This has the advantage of running faster because the robot performs only one transformation per cycle, but this also means that the robot is often using old information to guide its movements.

The approach used to detect plant pots in the final version consisted of two phases. The first was an ad hoc filtering phase for identifying and ignoring parts of the environment which were definitely not plant pots. The second phase was a gradient-descent phase that tried to minimize the mean squared error between a fitted circle, and a group of points that were considered likely to compose a circle. The filtering phase was further broken down into two passes. The first pass identified jump discontinuities in the scan by computing the distance between adjacent points. If two points were farther apart than a hand-tuned parameter, then this was considered a break between "segments", where each segment represented a group of consecutive points in the laser scan.



This diagram depicts the first major phase of the plant pot detector. The distances between adjacent points are computed, and these are used to segment the laser scan. In this example, we see three segments: a "green" segment on the left, a "yellow" segment in the middle, and another "green" segment on the right. The point-to-point derivatives are then computed to identify the instantaneous change in slope at each point in the laser scan, and any segment that does not exhibit a curvature in a hand-tuned range is thrown out. In this example, the left segment does not pass because it is flat (curvature near zero), the middle segment passes (has a small positive curvature), and the right segment fails (has a negative curvature).

The logic behind this was that if objects were spatially far apart in the environment, the distance between consecutive two points would be large, and we could safely break our segment there. If the distance between two consecutive points was small, then we could be reasonably certain that these two points belonged to the same physical object, and such should be in the same segment. Note that this could incorrectly throw out plant pots that were very far away from the robot. However, this was not a problem in practice, and could reliably detect pots up to 4 meters away.

Once the scan was broken down into segments, the second pass computed the "derivative" of each segment. If we think of each segment as a mathematical function, we can find the slope of the line defined by each pair of consecutive points. If the slopes were not monotonically increasing, and within an empirically determined range, then we could be certain that they were not part of a plant pot, and throw out the segment. This is because each plant pot is circular, and the slopes of short chords of the circle monotonically increase as one moves from left to right from the perspective of the robot.

The gradient-descent phase of the plant pot detector operated by iteratively fitting a circle to each remaining segment of the laser scan, and moving the x,y location of the center of the circle to minimize the mean squared error. If the MSE was found to be less than a threshold parameter, we stored the final circle as a potential plant pot. Finally, the center of the circle with the best fit was selected as the location of the plant pot.

This approach required a number of hand-tuned parameters, which took a significant amount of time to set properly. The difficulty to selecting good values was the result of noise in the laser scan (often just a few millimeters, but still significant, and of the cluttered environment we were trying to operate in. Finally, while the gradient-descent approach was faster than the Hough-transform by a factor of about 100, it could have been improved somewhat. A better solution might have been to use regression to a circle of a specified radius. However, we could not readily find a formula for such a regression, and efforts at deriving one were unsuccessful. We are aware of a linear, area-based approach to regression that may work, but haven't fully investigated this solution.

The final implementation was extremely accurate, functioning out to at least 4 meters. We could not test at distances farther than 4 meters because of problems with the laser mount: as received from the manufacturer, the laser pointed several degrees above horizontal, and it was unclear if this was a problem with the mounting bracket itself, or with the robot. This caused the resulting plant pot profile to be ellipsoid in the laser scan, rather than a perfect circle. In addition, if the robot was more than 4 meters away, the laser beams would extend above the pot, and nothing could be detected. We made numerous attempts to re-level the robot, and to shim up the laser mount, but these found only transient success. Regardless, the approach described above was accurate enough to correctly identify the presence and position of the plant pot upwards of 99% of the time, and was robust to sensor noise, partial occlusion, and an overall cluttered environment. Occasionally, people's legs and wastepaper baskets with rounded corners would show up as plant pots from far away, but in these cases, humans looking at the laser scan data would have arrived at the same conclusion.

Maintenance Bay Detection

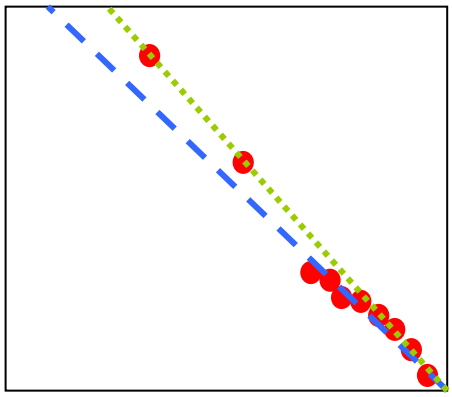
The maintenance bay's signature on the laser range finder was more complicated than that of a plant pot. Instead of a single circle, the bay showed up as three lines: two parallel, longer lines of a given length (the sides), and a shorter line connecting the endpoints of the other two lines (the rear). This observation breaks the maintenance bay detection problem into a line-detection problem, and a higher-level understanding problem, which functions by deciding which lines correspond to the bay.

The maintenance bay line detector went through three major attempts. The first was a Hough transform approach similar to that of the plant pot detector. However, instead of searching for circles in a Hough-space parameterized by the center of the pot's x,y coordinates, we searched for lines in a Hough-space parameterized by ρ, θ, as in the parametric formula for a line:

$$X \cos \theta + Y \sin \theta = \rho$$

Because of the trigonometric functions above, each point detected by the laser range finder is mapped to a sine wave in Hough-space. However, extracting the best line fit proceeds exactly as it does in the circle example described above. It is also subject to the same shortcomings: large memory and computational complexity, and difficulties finding the best granularity at which to perform the transform.

The second attempt was based on linear regression. We first performed a segmentation process similar to that done in the final approach for plant pot finding to reduce the laser scan into potential lines. Then we computed a regression for each segment, and kept all segments whose associate lines had a sufficiently small mean squared error. This method worked well in general, but had significant trouble with long lines that were close to the robot due to the radial spread of the laser beams. This caused a disproportionately large number of points to be sensed close to the robot, and very few sensed further away. When it came time to perform the regression, the closer points dominated the regression, and the resulting line was often a poor fit.



An illustration of conditions leading to the failure of the linear regression method for extracting lines. Due to the radial spread of laser beams, many points near the robot are seen, emphasizing a minor defect in the surface of the wall. The mean squared error regression for this data (blue line) does not necessarily follow the wall correctly (green line).

The final approach used was a computer vision line detection technique based on recursive subdivision. At each level of recursion, we assumed that the laser scan was a single, long line. We then found the point that deviated the most from that line, broke the line into two at that point, and recurred on both halves of the scan. This process terminates when the deviation between the most offending point and the line is below a hand-tuned threshold. The final version was very fast, on the order of a couple dozen milliseconds for a cluttered environment.

Once lines were extracted from the laser scan, the system used a case-based analysis to handle several different conditions which dictate which types of lines signify the presence of the maintenance bay. For example, if the robot was facing straight into the bay, it would see two parallel lines and a shorter line connecting the two, as described above. However, if the robot was to the right of the bay, it would not see the full length of the further bay wall. The final version was able to handle all such cases for the maintenance bay, and could detect the bay with well above 99% accuracy.

Aside from the initial Hough transform approach to line detection, the maintenance bay detection code was written and tested by Nathan Ratliff, an undergraduate at the University of Washington. Nathan also wrote, tested, and debugged the code for the potential field used in the behavior-based docking system.

The Completed System

The PlantCare project was completed in the summer of 2002, and video was taken to document this success. In this video, an iPaq was used to simulate a plant sensor. A low water signal was sent to a desktop computer, which then instructed the robot to water a particular plant. The robot complied, and returned to its maintenance bay successfully. The iPaq was used in lieu of a real plant sensor because of the necessity of causing a low water signal at will for the purposes of taping the event.

The entire project was implemented and functional, with the notable exception of the robot recharging the sensors. In theory, the only additional work needed to make this work would have been to turn the robot 180° after watering a plant, and "wiggle" the robot left and right until the inductive coil on the robot and that on the mote lined up. In order to maximize charging efficiency, the two coils should be parallel, and very close to each other. This would likely pose some difficulty in implementation, and due to lack of time and hardware difficulties, this aspect of the project was dropped.

This left the robot's portion of the project to be broken down into three areas: driving to an arbitrary point in the environment, docking with a plant, and docking with the maintenance bay. The first area was accomplished by the gradient-method navigation and localization modules acquired from ActivMedia. Because this was prepackaged code, we did not perform targeted empirical studies to determine how successful this code was. An estimate of how often this code worked would probably be upwards of 95% of the time.

The second and third portions of the project were tested empirically. Docking with a plant was successful on 38 out of 40 trials. Starting points were chosen arbitrarily along an approximately 120° arc, 1-3 meters away from the target plant. Interestingly, on each success, the robot did not drive directly to the plant. Instead, it missed the plant, having turned too far to the left on its initial approach. The robot then turned around, following the "swirl" of the vector field, and docked successfully on the second approach. This strange behavior is not well understood, but "emerged" over the course of a weekend. On a Friday, the robot docked 10 out of 10 times successfully, without this extra "swirl" behavior. By Sunday, the robot exhibited the swirl behavior approximately 25% of the time, and by the middle of the week, it was exhibited 100% of the time. We were authoritatively able to determine that the code running on the robot had not changed, and verified that the tire pressure was correct. However, we were never able to determine what caused the emergence of this extra swirl – we theorize that it may be due to gradual damage to the robot due to excess weight.

Docking with the maintenance bay was successful on 39 out of 40 trials. Starting points were chosen in the same manner used in testing the plant docking system. Even more interestingly, docking with the bay did not exhibit this swirling behavior, even though the potential field code used to guide the behavior was the same as that used for the plant docking system.

Unfortunately, the system was never fully deployed, and was retired soon after the taping. This was due to several factors, including persistent hardware difficulties with the robot, the time commitment that the project had already used, and the extra manpower that would be required to watch the system in case of failure. Because of this we were unable to produce MTBF statistics of the final system.

Hardware Difficulties

As in any robotics project, there were a number of issues that arose due at least in part to hardware. Towards the end of this project, I would estimate that a full one third to one half of my time was spend wrestling with hardware issues: maintaining the robot, conferring with ActivMedia on repair strategies, testing, etc. The major problem was stress on the robot from the excess weight that it was carrying – the laptop, the sometimes-full reservoir, the 4.5kg laser range finder, water pump inductive coils and rail systems. The Pioneer 2-Dx and 2-Dxe model robots are rated to carry 22kg, and we were careful not to go beyond this limit. However, after several months of carrying this load, the robots showed several signs of stress. Over time, the motors became sluggish, the axles started to exhibit a slight bend, and the ends of the axles started to become worn down by the brushes connecting them to the wheels.

The next issue that strongly affected the robot was a firmware upgrade for the robot provided by ActivMedia. One of the many upgrades issued during the project was an improvement to the PID controllers that govern the low-level movement of the robot, which lead to smoother robot control. However, there seemed to be a loss of torque associated with this patch. It is difficult to determine if this truly happened, because of the simultaneous effects from weight on the robot. However, it was obvious that the robot over time became unable to navigate some ramps in the lab. Hand tweaking the PID control values helped this somewhat. A related issue was that this upgrade was somewhat buggy. In addition to improving the PID controllers, it also deactivated the robot's stalling behavior. When the robot collides with an obstacle, it senses that it is providing the motors with substantial current, but is not moving. When this happens, the robot stalls the motors and goes limp for approximately one second, to prevent permanent damage to the robot's motor control boards. However, with this behavior disabled, this extra safety precaution was no longer in operation. We believe that it is possible that the robot was somewhat damaged by failed docking attempts and incidental collisions in the period of time between the upgrade was made and the bug discovered. In fact, this bug permanently damaged the motor control boards on our 2-Dx robot, loaned to us by Dieter Fox: ActivMedia's technical support asked us to test the robot's stalling behavior while the bug was still active. After some time, the board failed under the strain. The company was nice enough to promptly send replacement parts, as their testers experienced the same problem that night.

In the process of maintaining the robot, more than once I was required to remove the robot's wheels. However, replacing them accurately was quite difficult. There was no marking on the axle to specify exactly how far from the chassis the wheels should be located. If they were replaced too close to the chassis, the robot would turn faster, throwing off encoder information. If they were placed too far out from the chassis, the robot would then turn slower, reducing encoder accuracy, but in the opposite direction. Another interesting problem was that of tire inflation. The tires on the 2-Dx robot were foam rubber, and did not change shape except due to pressure. The tires on the 2-Dxe robot (used for our second implementation) were rubber tires that had to be periodically pumped up. While ActivMedia stated that they found the tires to work well under a variety of different tire pressures, we took special care to ensure that our tires were inflated to their recommended pressure.

The vast majority of the floors in the Intel Research Lab were covered with small pile carpeting. While this may not sound like a major obstacle to research robotics, the Pioneer's wheels experienced wheel-slippage while driving on this surface. In practice, this means that the distance the robot actually traveled often came up shorter than indicated, up to 10% shorter when compared to operation on flat, smooth surfaces. Because our Saphira/Aria implementation was mostly behavior-based, this did not cause many problems except for the "scripted" portions of the docking phases. However, this would cause serious difficulties for the BeeSoft implementation, because the entire docking phases were scripted.

As mentioned previously in the feature detection section, we had significant problems with the laser mount. Despite the fact that the laser was mounted to the manufacturer's specifications, the laser still was tilted upward, causing the beams to travel over plant pots that were further than 4 meters away. This also caused problems in that the plant pot's image on the laser range finder was now ellipsoid, not circular, and the radii of this ellipsoid depended on how far the pot was from the robot. While the final detector was robust

enough to handle this type of sensor noise, significant time was spent tuning the detector and looking for hardware remedies to this problem.

The environment also posed a number of hazards for the robot. "Natural" obstacles such as chairs were bountiful. Office chairs are to robots what icebergs are to cruise ships. Traditionally, only the top 10% of an iceberg is visible above the surface of the water, while the submerged remainder can damage a vessel, even if it is far away from the apparent location of the iceberg. Similarly, office chairs have a large footprint below the level of the laser range finder; the robot can only see the main support post, which typically has a diameter of about 2 inches. This means that a robot can easily catch itself on the wheels of such a chair. This happened numerous times when the robot was driving around the lab autonomously. Humans also pose an interesting hazard for robots. Visitors, children, and other people who are not used to working in an environment with a robot often want to "play" with it, often testing its obstacle avoidance abilities. While neither of these damages the robot, they do underscore the difficulties in working in a real-life office environment.

Future Project Extensions

There are many directions this project can take, should it be revived.

- **Automatic mapping**: As previously mentioned, the robot had to be manually driven by joystick around the lab in order to produce the map. One enhancement that would better meet the goal of zero-configuration would be to make the robot decide what parts of the lab were to be explored, and to guide its own mapping. Efficient mapping strategies currently exist that would make this feasible, such as FastSLAM *[Montemerlo, Thrun, et. al.-2002]*.
- **Automatic plant and maintenance bay discovery**: In a perfect zero-configuration system, the robot would be able to locate new plants, and detect when plants were removed or move around. There are several ways to accomplish this. First, the robot could be programmed to drive around the lab when it is not servicing plants and motes. When it locates an object via the laser range finder that looks like a plant pot, it could gently bump into the object. An accelerometer could be attached to each mote, which would then provide feedback to the robot, sending a message such as "plant number 157 has been bumped". It is also possible that the 802.11 wireless network could be used in some way to triangulate the position of new plants, to limit the initial search space.
- **Mote recharging**: The original plan for this system called for the robot to recharge motes that were running low on power. Due to insufficient time and uncooperative hardware, I was unable to complete this task. In theory, it would require only simple changes to the standard plant docking procedure, though of course, things do not always proceed according to plan. Completing this extension would provide strong evidence for the future of robots in wireless sensor networks and ubiquitous computing.
- **Sensor fusion**: Current experiments at the Intel Research Lab are focusing on using a variety of fixed sensors to track people's day-to-day activities, including infra-red badges, ultrasound sonar beacons, and fixed laser range finders. The PlantCare robot's localization could conceivably be improved by added equipment to interface with these tracking technologies, and finding sensor fusion strategies to incorporate these with the particle-filter localization module.
- **Human-Robot Interaction**: Human-computer interaction is a rich field in computer science research, and Human-Robot interaction promises to be even richer. *[Montemerlo, Pineau, et. al.-2002]* Aside from more tradition issues of interface design (graphical/mouse/keyboard, voice-activated), robots offer more opportunities for giving feedback to the user in the form of movement. Robots also provide new methods for interfaces, such as humans teaching robots by example (such as placing objects in a robot's gripper), or physically demonstrating a task that the robot should learn on its own. More benign issues are also possible, such as planning routes to avoid high-traffic regions and times, or interfacing with external people-tracking technologies as mentioned above to plan paths that avoid inconveniencing humans or other robots in the environment.

Acknowledgements

I want to thank Dieter Fox and Gaetano Borriello for their advice and generous funding on the PlantCare project. I also received tremendous support from Anthony LaMarca, David Koizumi, Ken Smith, Matt Lease, Nathan Ratliff, Stefan Sigurdsson, and Waylon Brunette. And special thanks to the Intel Research Lab, for providing an environment, a robot, and materials for this work.

References

ActivMedia Robotics, http://www.activmedia.com/

Fast Light Toolkit, http://www.fltk.org/

D. Fox, W. Burgard, and S. Thrun., *Markov Localization for Mobile Robots in Dynamic Environments*, Journal of Artificial Intelligence Research (JAIR), 11, 1999.

Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, Kristofer Pister, "System architecture directions for network sensors." In the Proceedings of ASPLOS, 2000.

Luca Iocchi, Daniele Nardi**,** *Hough Transform based Localization for Mobile Robots***,** Advances in Intelligent Systems and Computer Science, World Scientific Engineering Society, 1999.

Kurt Konolige, *A Gradient Method for Realtime Robot Control*, IROS-2000.

Anthony LaMarca, et. al. *PlantCare: An Investigation in Practical Ubiquitous Systems*, 4th International Conference on Ubiquitous Computing 2002.

Anthony LaMarca, et al. *Making Sensor Networks Practical with Robots*, International Conference on Pervasive Computing 2002.

M. Montemerlo, S., Thrun, D. Koller, B. Wegbreit, FastSLAM*: A Factors Solution to the Simultaneous Mapping and Localization Problem*, AAAI-2002.

M. Montemerlo, J. Pineau, N. Roy, S. Thrun, and V. Verma, *Experiences with a Mobile Robotic Guide for the Elderly*, AAAI-2002.

Kevin Sikorski**,** *Sonar-based Robot Navigation Using Occupancy Grids***,** undergraduate honors thesis, Brown University, 1999.